What Are HDF5 Things?

Dana Robinson (THG)

# *Things* You Need To Model In Your VOL Driver

**NOTE:** This document describes the abstract HDF5 data model. It does NOT describe peculiarities of any storage modality or platform. Any real-world implementation of the data model, including The HDF Group's canonical HDF5 library, will have limitations which are not addressed here.

## Files

A file is a container which holds objects. Every file contains a root group, which is the beginning of the file's hierarchical storage layout.

Every file has a name, which is used to access the file.

## Objects

Objects are the fundamental things that are stored in a file. Datasets, groups, and (committed) datatypes are the only classes of objects. Note especially that attributes are NOT objects. Also note that, formally, HDF5 objects do not have names; it is the links that bear the names.

Another way of thinking about this is that an HDF5 object is anything that can be a target of a link.

## Datasets

A dataset is an *n*-dimensional array of data of a particular datatype. Each element in a dataset has the same datatype, though these datatypes may be complex. Datasets are not ragged; each dataset is fully extended in each of the *n* dimensions.

Each dataset must store its own datatype. This inherent datatype is accessed via dataset API calls and is not necessarily an object in the file (though temporary, in-memory type representations could be created by the implementation and returned to the user via this mechanism). A dataset also has a fill value, which is the value that is returned if no data have been written to that region of the dataset. The datatype of a dataset is not mutable.

## Groups

A group is a collection of links.

## Links

A link is a uni-directional connection between a group and a destination object (or target). Cycles are allowed. The representation of the link is not important for the data model (i.e.: the data model does not distinguish between low-level hard links and stored-path soft links). Links may be between objects in the same file or in different files. Links to objects stored in different files are called external links.

A link has a name, which is a character string used to access the link, or, more informally, the destination object. A link name may represent a single link or a traversal along multiple links. In the latter case, the link names must be separated by the '/' path separator character. The special character '.' refers to the current group. A link path that does not begin with a path separator is interpreted as relative to the current location in the file hierarchy. A link path that beings with a path separator is interpreted as an absolute path which begins at the file's root group.

An external link also stores a file name, which is used to access the file in which the link target resides.

Link targets do not have any way to access the links which point to them, so it is not possible to traverse a file in the reverse direction.

## Attributes

An attribute is a name-value pair attached directly to an object. They are not the target of links in the file and are hence not objects. The name is a string, which is used to access the attribute. The value can be any datatype.

Each attribute must store its own datatype. This inherent datatype is accessed via attribute API calls and is not necessarily an object in the file (though temporary, in-memory type representations can be created by the library and returned to the user via this mechanism). The datatype of an attribute is not mutable.

## Datatypes

Datasets and attributes can have a wide variety of types. Types can be implicit and stored with datasets and attributes. A datatype can also be stored in a file as the target of a link on its own, which is called a named or committed datatype.

The implicit datatypes associated with datasets and attributes are not necessarily file objects. They are accessed via the dataset or attribute API calls and are not the targets of links. On the other hand, a stored (or, more accurately, *committed*) datatype is an object and is accessed via a link.

### Atomic Datatypes[[1]](#footnote-1)

|  |  |
| --- | --- |
| Bitfield | An arbitrary number of bits which can be individually set |
| String | A character string in an arbitrary encoding |
| Object Reference | A reference to an object in an HDF5 file |
| Dataset Region Reference | A reference to a region of arbitrary complexity in an HDF5 dataset |
| Integer | An integer number of arbitrary size |
| Floating-Point | A floating-point number of arbitrary size and precision |
| Opaque | The bytes of an opaque type are not interpreted by the data model. An optional string (called a *tag*) may be added to the opaque type (this may help downstream software/users interpret the bytes). |

### Composite Datatypes

|  |  |
| --- | --- |
| Array | Contains a fixed number of elements of an arbitrary datatype |
| Variable-Length | Contains a variable number of elements of an arbitrary datatype |
| Compound | Each element contains a number of named fields of specified (arbitrary) datatype. The name of each field must be unique in the type. Each field does not have to be of the same datatype. |
| Enumerations | Values are be members of key-value pairs. The keys are names, which must be unique in the type. The values are of integer type, and must be unique in the type[[2]](#footnote-2). |

# *Operations* On Things You Need To Model In Your VOL Driver

## Files

## Objects

## Datasets

## Groups

## Links

## Attributes

## Datatypes

# Other Stuff

## Error API

## Identifiers

## Property Lists

## Plugin API

## Dataspaces

## Filters

1. Technically, there’s an atomic “time” datatype also, but it is deprecated. [↑](#footnote-ref-1)
2. Note that this is more restrictive than enumerated types in most programming languages. [↑](#footnote-ref-2)